Nonautonomous elementary net systems and their application to programmable logic control

Brusey, J. , McFarlane, D.C. and Thorne, A.

Author post-print (accepted) deposited in CURVE March 2012

Original citation & hyperlink:
Brusey, J. , McFarlane, D.C. and Thorne, A. (2008) Nonautonomous elementary net systems and their application to programmable logic control. IEEE Transactions on Systems, Man and Cybernetics. Part A: Systems and Humans, volume 38 (2): 397-409.
http://dx.doi.org/10.1109/TSMCA.2007.914775

Publisher statement: © 2008 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Copyright © and Moral Rights are retained by the author(s) and/ or other copyright owners. A copy can be downloaded for personal non-commercial research or study, without prior permission or charge. This item cannot be reproduced or quoted extensively from without first obtaining permission in writing from the copyright holder(s). The content must not be changed in any way or sold commercially in any format or medium without the formal permission of the copyright holders.

This document is the author's post-print version, incorporating any revisions agreed during the peer-review process. Some differences between the published version and this version may remain and you are advised to consult the published version if you wish to cite from it.

CURVE is the Institutional Repository for Coventry University
http://curve.coventry.ac.uk/open

Non-Autonomous Elementary Net Systems and their Application to Programmable Logic Control

James Brusey, Duncan C. McFarlane, and Alan Thorne

***Abstract*— A novel approach to Petri net modelling of Programmable Logic Controller (PLC) programs is presented. The modelling approach is a simple extension of Elementary Net Systems and a graphical design tool that supports the use of this modelling approach is provided. A key characteristic of the model is that the binary sensory inputs and binary actuation outputs of the PLC are explicitly represented. This leads to two improvements: outputs are unambiguous, and interaction patterns are more clearly represented in the graphical form. Use of this modelling approach produces programs that are simple, light-weight, and portable. The approach is demonstrated by applying it to the development of a control module for a MonTech Positioning Station.**

***Index Terms*— Petri nets, programmable logic controllers**

I. Introduction

PETRI nets [1], [2] have long been used as a way of modelling manufacturing operations [3], [4]. In comparison with finite state automata [5], they have the advantage that they readily model processes that are sometimes concurrent and sometimes sequential. For example, individual parts that are combined together into a product might be machined separately and independently. These independent processes can proceed at any rate relative to each other. This means that there are many possible ways for the state of the whole system to evolve over time, depending on exactly when each machine reaches each different stage in its processing cycle. Even though the components are machined separately, since they come together during the final stages, a model of the complete system cannot merely be a collection of independent automata, but must represent, for example, the way these final stages must wait until all prior stages are complete. Petri nets can be used to express and reason about concurrent processes such as these in a simple and compact way [6], hence their popularity as modelling tools for the manufacturing domain.

Given that the Petri net is an appropriate model for a manufacturing operation, it also seems natural to derive the corresponding controller for the operation directly from this model. A *controller*, in this sense, means the program or set of programs that runs on a Programmable Logic Controller (PLC)—which is connected to sensors and actuators—allowing it to sense and change the manufacturing environment. An immediate concern is that there is often not sufficient information in the Petri net model to enable an unambiguous derivation of a controller. That is, the Petri net is too incomplete a description of the system. For example, details such as the physical layout of the shop floor, and the nature of the physical sensing and actuation configuration can generally not be expressed in a Petri net model. Previous researchers in this area, notably Chirn and McFarlane [7], Frey [8]–[13] and Uzam and Jones [14]–[17], have addressed the problem of integrating sensing and actuation by adding an *annotation* on each node of the net. Each annotation expresses the flow of information to or from the outside world associated with that node. Such Petri nets are generally classified as *non-autonomous* in the sense that they influence, and are influenced by, the external environment [1]. There are various forms of such nets depending on the type of annotation, such as Frey's signal interpreted Petri nets (SIPNs) [13]. In this paper, non-autonomous Petri net will be used to refer to a net that defines its external interface via an annotation for each node.

The problem with existing approaches to coding the behaviour of a PLC as a Petri net is that they treat inputs and outputs as being external to the state of the PLC [7]–[17]. This is at first a conceptual difficulty, meaning that the correspondence to a finite state automaton is not clear, for example. But it also causes some potential ambiguity in the resulting control implementation. This paper proposes that it is advantageous to treat inputs and outputs as part of the controller state, and to represent them explicitly in the Petri net model. There are two advantages. First, it then becomes impossible to set outputs in an ambiguous way. It is not possible (as it is in the SIPN model) to specify, for example, that a certain output signal is to be set both on *and* off simultaneously [9]. Second, inputs become a structural element and thus interaction patterns are shown more clearly.

This paper shows how a PLC program can be compiled directly from a Petri net *without* requiring annotations. This is referred to as *compiling* the net, in the sense that a high-level, abstract representation of the behaviour of the program is being used to automatically and directly generate executable PLC code. A graphical tool that supports this process is a central outcome of this research. The general aim of such a tool is to ease the burden on the engineer. This overall aim resolves into four fundamental requirements:

1) To *correctly* translate the Petri net behaviour into corresponding program behaviour,
2) To *automatically* perform the translation without requiring that the engineer ensures boundedness, absence of conflict, or absence of output ambiguity,
3) To produce a *variety of output forms* including ladder logic diagrams,
4) To support code reuse through *modularity*.

This work was supported by Auto-ID Labs, I*PROMS, and the Cambridge-MIT Institute.

0000–0000/00$00.00 © 2004 IEEE

The paper is organised as follows. The next section reviews related work including a brief introduction to the theory of Elementary Net Systems. This theoretical basis is then used to form a new class of net, referred to in this paper as Non-Autonomous Elementary Net systems. Section III shows how the model can be automatically translated into a form suitable for PLCs and proves that the resulting code has equivalent behaviour. Section IV discusses several aspects of the use of this approach, particularly focusing on coordination and modularity. Section V demonstrates the compilation method by applying it to a real device control problem. Finally, conclusions are presented in the last section.

II. Related Work

A basic approach to translating Petri net behaviour into ladder logic code is suggested by Chirn and McFarlane [7]. Their approach is easy to understand and perform, however it may produce some unwanted side-effects where a token is temporarily in two places at once. Usually, this sort of side-effect has no impact on the overall behaviour. The difficulty is that it *may* have an impact, and in particular that the problem will tend to be hard to diagnose because the situation occurs for only a single PLC cycle.

A number of other authors have developed approaches to producing PLC programs from Petri nets [12], [13], [16], [18], [19]. The two most well known ones are found in theses by Frey [13] (referred to here as the signal interpreted Petri net or SIPN approach) and Uzam [15] (referred to as the token passing ladder logic or TPLL approach). Frey and Minas [11] have also built a graphical development tool to support their approach [20]. Using their tool, the resultant nets can be simulated and analysed for liveness and reachability. In their approach, IEC 61131-3 standard Instruction Lists (IL) are produced by the compilation process (Frey has also examined generating sequential function charts (SFC) [9]). The approach here is similar, but whereas Frey and Minas's tool produces statements that include flow control instructions, the approach presented here does not. Avoiding flow control instructions should make the program easier to debug because it means that every statement is executed during every scan cycle. In addition, the lack of flow control instructions means that the program is readable in ladder diagram form, thus providing an easier migration path for manufacturing environments where the use of ladder diagrams is entrenched.

Lee *et al.* [19] propose a method for deriving ladder logic from Control Petri Nets. Control Petri Nets (or CPNs) add enabling and inhibiting arcs. Lee *et al.*'s approach is to assume that the engineer has produced 1-safe nets that have certain restrictions about what conflicts can occur. Their approach makes the optimisation of combining the calculation of which transitions to fire and the calculation of the updated marking into a single step. However this appears to produce incorrect results for some nets. See the appendix for a counter-example.

Correctly dealing with concurrent firing of transitions is discussed by Frey [8]. Frey uses a similar approach to the one described here, that of updating tokens after determining that a transition should fire. Conflicts are avoided by firing transitions sequentially. In contrast, the approach presented in this paper allows some simultaneous firing by checking for potentially conflicting transitions and ensuring that they do not fire simultaneously.

A key issue with non-autonomous PN-based approaches is that there is not a one-to-one correspondence between outputs and a subset of the places in the net. Therefore it is possible that when many places are marked, more than one of these places will set the value of the same output, and these various settings may be inconsistent. Minas and Frey [21] mention that this was a difficulty that their students frequently encountered while using their SIPN-based tool. Having binary actuator outputs correspond to a subset of places in the Petri net prevents this ambiguity. Then, if two active "processes" within the net attempt to turn on an output, one will "block", or wait until the other has finished.

As with the work here, Frey's and Chirn's approaches are based on binary marks. In contrast, Uzam and Jones [14]–[16], [22] developed the Token Passing Ladder Logic (TPLL) method that supports multiple tokens per place. As Peterson [2] notes, Petri's original model involved binary marks and simple arcs, whereas this has been shown to be too limited for some problems and has been extended to allow multiple tokens per place, and multiple arcs between any two nodes. Nevertheless, Elementary Net systems, with binary marks and simple arcs, have a more natural correspondence to the binary sensors and actuators controlled by a PLC. In addition, while place-transition nets theoretically have no limit on the tokens per place, when this is translated to a PLC it would seem necessary to include a limit to avoid overflow errors. TPLL does not seem to include such a limit.

Jones *et al.* [22] provide an approach for incorporating time delays within the resulting program. However, their approach potentially leads to problems both of overflow and of underflow in the number of tokens per place.

Boucher [18] mentions the use of Petri net places as a basis for communicating resource locks between systems. In particular, Boucher shows how handshaking between two systems can be performed. However this is merely shown as a way of modelling the communication protocol, and the implementation details, such as ensuring that shared data areas (if any) are used in a safe manner, are not discussed.

Feldmann *et al.* [23] define an extension to coloured Petri nets called Ordered Coloured Petri Nets (or OCPNs) and show how they can be translated to IEC 61131 Structured Text (ST). The OCPN approach is more sophisticated again than Uzam's TPLL, since not only may there be multiple tokens per place, but also the tokens are coloured. Each transition is annotated with a guard function that is a conjunction of boolean variables. Also each arc is annotated by one of several standard colour functions.

A difficulty with many of the existing approaches is that, with the exception of Frey and Minas's compiler [8], they are largely manual. In their review of this subject area, Peng and Zhou [24] note the need for verified, automatic conversion between Petri net and IEC 61131-3 programming languages. They also note the need for effective compositional methods for modular, distributed control systems, and in particular note

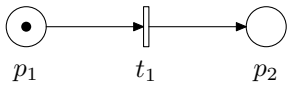

Fig. 1. A simple EN system corresponding to $P = \{p_1, p_2\}$, $T = \{t_1\}$, $F = \{(p_1, t_1), (t_1, p_2)\}$, $C_{\text{in}} = \{p_1\}$.

the importance of interfaces or interlocks between different PN-based control modules. This paper aims to address these needs with a minimalist model that is thus hopefully easily understood.

To describe this approach rigorously it is first necessary to review the theory of elementary net systems on which the approach is based.

A. Elementary Net Systems

An Elementary Net (EN) system [25] is a fundamental form of Petri net. In comparison with ordinary Place / Transition systems (P/T systems) [25], EN systems only allow a single token per place, whereas ordinary P/T systems allow any number. Condition / Event (C/E) Systems also have only one token per place and form the basis for SIPN systems [13]. The difference between C/E and EN systems is quite subtle and the terms are often used interchangeably. Pomello and Bernardinello [26] define the difference as being that with C/E systems forward and backward reachable cases are considered, whereas for EN systems, there is an initial configuration and the system only evolves in a forward direction.

The particular properties of EN systems can be described as follows. An elementary net system is a 4-tuple, $\mathcal{N} = (P, T, F, C_{\text{in}})$, where (P, T, F) is the underlying network consisting of a set of places or states P, a set of transitions or events T and a set of relations or directed arcs F. Transitions are distinct from places $P \cap T = \varnothing$ and directed arcs join either a place to a transition or a transition to a place $F \subseteq (P \times T) \cup (T \times P)$. It is usual to represent the net diagrammatically with circles for places, boxes for transitions, and lines with arrows for directed arcs as shown in figure 1. A *configuration* $C \subseteq P$ corresponds to the dynamic state of the net and is initially C_{in}. The configuration C is represented graphically by the presence of a token (i.e. a large dot) in places that are in C. Roughly speaking, a place is *marked* if it is in C or *unmarked* otherwise. For each $x \in P \cup T$, ${}^\bullet x = \{y \in P \cup T : (y, x) \in F\}$ is the pre-set or set of elements with arcs leading into x, while $x^\bullet = \{y \in P \cup T : (x, y) \in F\}$ is the post-set or set of elements with arcs leading from x. That is for any node x, be it a place or transition, its pre-set ${}^\bullet x$ contains all nodes that have arcs directed to it, while its post-set $x^\bullet$ contains all nodes that have arcs directed away from it. For a set of nodes $X \subseteq P \cup T$, its pre-set ${}^\bullet X = \bigcup_{x \in X} {}^\bullet x$ is the union of pre-sets of its elements and similarly its post-set $X^\bullet = \bigcup_{x \in X} x^\bullet$ is the union of post-sets of its elements.

The state of the net changes by "playing the token game". This game has a simple set of rules [25, page 32], which is restated here.

Definition 1: The configuration C of an EN system $\mathcal{N}$ changes according to three rules:

1) A transition t is *enabled* to fire at configuration C, denoted $C\,[t\rangle$, if all of its pre-conditions are marked ${}^\bullet t \subseteq C$ and all of its post-conditions are unmarked $t^\bullet \cap C = \varnothing$.
2) When a transition t fires at C, tokens are removed from all pre-conditions and added to all post-conditions, yielding a new configuration $C' = (C \cup t^\bullet) - {}^\bullet t$. That C' is the result of t firing at C is denoted as $C\,[t\rangle\,C'$.
3) A set of transitions $U \subseteq T$ is a step enabled at C, denoted $C\,[U\rangle$, iff (a) all transitions in U are enabled at C, and (b) for all $t_1, t_2 \in U$ with $t_1 \neq t_2$, ${}^\bullet t_1 \cap {}^\bullet t_2 = \varnothing$ and $t_1^\bullet \cap t_2^\bullet = \varnothing$.

The latter condition of the third rule ensures that the firing of any individual transition in U does not affect whether any other transition in U is enabled. Note that, in an EN system, if some preconditions are also postconditions, or in other words, if for any transition $t \in T$, ${}^\bullet t \cap t^\bullet \neq \varnothing$, then the transition t can never be enabled. EN systems that have no such transitions are referred to as *pure*.

This section has introduced the domain. In the following section, a non-autonomous EN system model is defined and the mapping to PLC code described.

III. Non-Autonomous Elementary Nets

EN systems are autonomous. Programmable Logic Controllers (PLCs) are not, since they must interact with the world. The behaviour of a PLC is much simpler, though, than a general purpose computer. They operate in a cyclic fashion made up of three phases. First, sensor values are read into a special memory area reserved for inputs (that is otherwise read-only). Second, the user's program is executed exactly once. Third, actuator states are updated based on the values in the memory area reserved for outputs. These three phases are repeated indefinitely.

PLC programs are usually written in an intermediate level language, such as ladder diagrams (LD) or function block diagrams (FBD), that have a close correspondence with the low-level machine instructions. Of the standard PLC languages, as defined by IEC 61131-3 [27], the Instruction List (IL) language has the closest correspondence, and is roughly equivalent to assembly language. All LD programs can be expressed as IL, but not all IL programs can be expressed in LD.

The non-autonomous EN system model of a PLC to be developed here is based on the observation that elementary net places are similar to the binary inputs, outputs and (internal) data storage areas of a programmable logic controller. Specifically, the set of places for an elementary net are equivalent to the bitwise address locations for binary sensors (inputs), binary actuators (outputs) and binary data storage locations. The configuration of the Petri net then corresponds to which bit address locations are "on" (i.e. set to "1"). This is a useful analogy because it allows the dynamic behaviour of a PLC to be modelled as an Elementary Net system.

Of the three basic types of PLC register locations (inputs, outputs and data storage), inputs are somewhat exceptional, in that they are not directly affected by the PLC's operation.

Past work, such as that of Frey *et al.* [8], has resolved this by altering the semantics of transition firing to additionally check an associated boolean expression, and to only fire if the expression is true. In comparison, in the approach presented here, inputs are integrated into the Elementary Net system by having an input correspond to a place. Nevertheless, it is necessary to augment the basic EN system model to include a special type of place $i \in \mathcal{I}$ to represent PLC inputs, where $\mathcal{I} \subseteq P$. Such input places must be treated differently because they can only be affected by events that are external to the modelled system. For example, a light sensor will only be affected by the presence of light, and not the internal state of the PLC.

Previous work [7], [8], [14] has treated the state of *outputs* as external to the net. In the approach presented here, each output, such as a valve, solenoid, or light, is represented as a *place*. Rather than turning on or off outputs on entry to a place (as in [7], [8], [14]), in a non-autonomous EN system a single output is turned on when a token enters the place corresponding to that output, and is turned off when the token leaves that place. Simply stated, the state of a single output corresponds to the state of a single place. An advantage of this approach is that it means that there can be no ambiguity in how the output is set, since there is only one place corresponding to each output. A drawback, however, is that outputs must be binary. It would be possible to support other types of outputs by allowing annotation on places.

Remark 1: In summary, given the three distinct sets of inputs $\mathcal{I}$, outputs $\mathcal{O}$, and data storage $\mathcal{D}$, the set of non-autonomous EN system places is the union of all three $P = \mathcal{D} \cup \mathcal{I} \cup \mathcal{O}$.

Note that there is a mapping (or *injection*) from non-autonomous EN system places to PLC address locations. That is, each non-autonomous EN system place must correspond to a unique bit address in the PLC, and this can be a data storage address, an input address, or an output address. Only input places can correspond to input addresses. It is up to the programmer to provide the mapping of places to bit addresses. Transitions also must be mapped to data storage bit addresses, but this can usually be done automatically. Note that all places and transitions are mapped to unique addresses but the inverse mapping from addresses to places or transitions may only apply to a subset of all possible addresses.

A. Defining Non-Autonomous EN Systems

As discussed above, the rules of the token game must be revised to suit this slightly altered model by subtracting the set $\mathcal{I}$ from all pre-conditions and post-conditions in rules 2 and 3 in Definition 1. To make it distinct, the resulting model is referred to as a *Non-Autonomous* Elementary Net system.

Definition 2: A Non-Autonomous Elementary Net system is a tuple $M = (P, T, F, \mathcal{I}, C_{\text{in}})$, where the tuple (P, T, F, C_{in}) corresponds to the underlying Elementary Net system whose behaviour is modified such that all places in $\mathcal{I} \subseteq P$, referred to as *input places*, remain unaltered by the firing of a transition. The initial configuration defines only the state of non-input places $C_{\text{in}} \subseteq P - \mathcal{I}$.

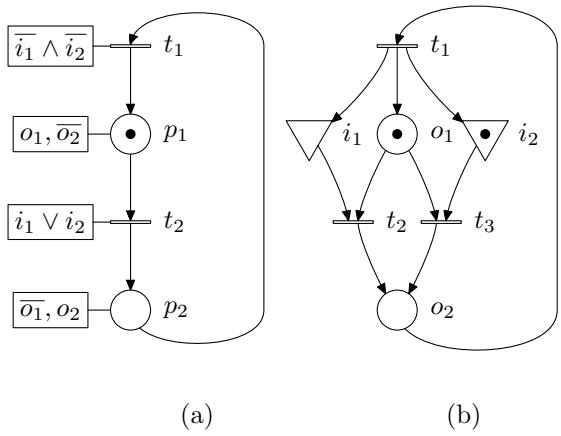

Fig. 2. Comparing (a) a non-autonomous Petri net with (b) the equivalent non-autonomous Elementary Net system. In the PN annotations, negation is represented by a bar, such as $\overline{i_1}$. In the EN, a triangle is used to denote an input place.

The rules for "playing the token game" with Non-Autonomous EN systems can be stated as follows.

Definition 3: The configuration C of a Non-Autonomous EN system M changes according to three rules:

1) A transition t is *enabled* to fire at configuration C if all of its pre-conditions are marked ${}^\bullet t \subseteq C$ and all of its post-conditions are unmarked $t^\bullet \cap C = \varnothing$.
2) When a transition t fires at C, tokens are removed from all non-input pre-conditions and added to all non-input post-conditions, yielding a new configuration $C' = (C \cup (t^\bullet - \mathcal{I})) - ({}^\bullet t - \mathcal{I})$.
3) A set of transitions $U \subseteq T$ is a step enabled at C iff (a) all transitions in U are enabled at C, and (b) for all $t_1, t_2 \in U$ with $t_1 \neq t_2$, ${}^\bullet t_1 \cap {}^\bullet t_2 \subseteq \mathcal{I}$ and $t_1^\bullet \cap t_2^\bullet \subseteq \mathcal{I}$.

This definition derives from Definition 1 and the requirement that the state of input places is unaffected by the process of firing a transition. To see how this modified net compares with traditional non-autonomous Petri nets [7]–[17], consider the simple example shown in figure 2. In this example, two motors o_1 and o_2 are controlled by inputs i_1 and i_2 so that when either switch is on, o_2 is turned on, and o_1 otherwise. The two motors should not be on at the same time. In the PN (figure 2(a)), inputs and outputs are distinct from places. Outputs can be set (or reset, if negated) on entry to a place by coding an annotation (represented by a labelled box connected to the place by a line). Similarly, inputs annotate transitions and thus are additional conditions for transition firing. For example, t_1 has the annotation $\overline{i_1} \wedge \overline{i_2}$. Therefore, i_1 and i_2 must both be off (because the terms are negated) for t_1 to fire (as well as p_1 being unmarked, and p_2 being marked). Place p_1 has the annotation $o_1, \overline{o_2}$ meaning that when it receives a mark, o_1 will be turned on and o_2 will be turned off. Note that any place can update any output and this may lead to some confusion. Although it is straightforward to check the behaviour of this net, determining which places affect a particular output may be more difficult with a larger net.

Now consider the Non-Autonomous Elementary Net system in figure 2(b). A triangle is used to denote an input place. Note

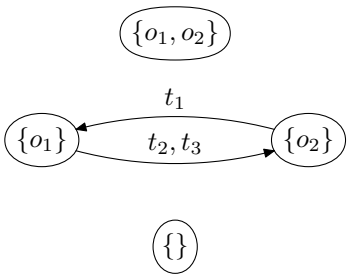

Fig. 3. The sequential configuration graph for the non-autonomous EN system in figure 2(b).

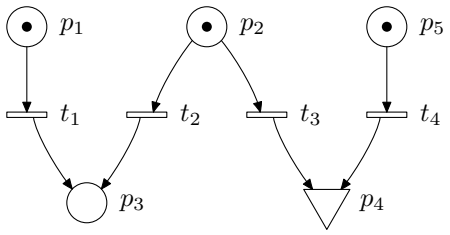

Fig. 4. Example of conflicting transitions. Transitions t_1 and t_2 are in conflict because if they both fired simultaneously p_3 would overflow. Similarly t_2 and t_3 are in conflict because if both fire at once, p_2 will underflow. Transition t_4 does not conflict with t_3 since the place that they have in common is an input place.

that there are no annotations and only the standard rules for firing apply. Therefore, t_1 can fire if o_2 is marked and i_1, i_2 and o_1 are unmarked. When it fires, o_1 becomes marked and o_2 becomes unmarked according to the standard rules, however, in contrast to the standard rules, i_1 and i_2 are unaffected. The inputs i_1 and i_2 will only be affected by the state of their associated sensors. In this form of net, all transitions that affect an output must be connected to it by an arc. Therefore, even in a large net, it should be easy to find all the ways in which an output can be changed.

Remark 2: It is not always possible to map 1-safe non-autonomous PNs to non-autonomous EN systems since there may be ambiguity in the setting of output places possible in the PN.

As the Petri net model has been altered, this leads to the question of how the static analysis of such altered Petri nets might change. For an EN system, it is possible to derive a sequential configuration graph (SCG) that has a node for each possible configuration (that is, each possible marking), and arcs corresponding to enabled transitions in those markings [25]. Analysis of properties such as *reachability* or *liveness* can then be derived from the SCG. Specifically, a configuration is reachable if, in the SCG, there is a path to it from the initial configuration. A transition is live if it can be eventually fired when starting from any reachable configuration.

The Non-Autonomous EN system also contains input places $i \in \mathcal{I}$, which have slightly different semantics (their configuration is not updated by transitions firing), and this necessarily affects the analysis. To model the possibility that the external environment could change in any possible way, arcs must also be included which allow any set of inputs to change their state at any time. Alternatively, input places can be disregarded completely. The following result shows that this simpler approach is equivalent for the purposes of identifying reachability.

Lemma 1: The sequential case graph (SCG) for the Non-Autonomous EN system $M = (P, T, F, C_{\text{in}}, \mathcal{I})$ with arcs added to allow input places $i \in \mathcal{I}$ to change their state at any time, is equivalent (in terms of reachability) to the SCG of the EN system $M' = (P - \mathcal{I}, T, F - \mathcal{I} \times T - T \times \mathcal{I}, C_{\text{in}})$.

Proof: See appendix. ■

The SCG for the Non-Autonomous EN system in figure 2(b) is given in figure 3. From this graph it is clear that, for example, configuration $\{o_1, o_2\}$ is not reachable, and so both motors can never be on simultaneously (assuming the initial configuration is $\{o_1\}$).

Having developed a theoretical basis, the process of converting Non-Autonomous EN systems is now described in the next two sections, starting with the process of compiling the arcs associated with transitions, and followed by the process of compiling the arcs associated with places.

B. Compiling Transitions

For the purposes of "playing the token game" with a PLC, transitions and places can be viewed as boolean variables. Based on the rules stated in Definition 1, a series of boolean statements can be formed to determine which transitions can fire as (representing logical AND by $\wedge$, OR by $\vee$, and NOT by $\neg$). In the simplest case, there is no conflict between transitions and a transition is enabled when all of the pre-conditions and none of the post-conditions are marked. Conflict can occur when two transitions have arcs to or from the same place. An example showing these two types of conflict is shown in figure 4. The possibility of two or more conflicting transitions firing simultaneously is precluded by rule 3 in definition 3. Nevertheless, the question remains of how to implement this restriction. If the restriction were ignored, the simultaneous firing of conflicting transitions would lead to places either overflowing (i.e. a transition firing when its post-condition already had a token), or underflowing (i.e. a transition firing when its pre-condition had already had its token removed).

Fortunately, it is not necessary to insert code into the PLC program to evaluate which transitions might be conflicting. Instead the compiler can simply look for the *possibility* of conflict between two transitions (also known as structural conflict), and guard against them firing simultaneously. To do this, an additional test is added to each update for a transition with the potential of conflict with a preceding transition. For example, the logical update for t_2 in figure 4 is $t_2 \leftarrow p_1 \wedge \neg p_3 \wedge \neg t_1$. Given that t_1 will be evaluated first, t_2 will never fire when t_1 is firing. Potential conflict can be rigorously defined as follows.

Definition 4: Let $M = (P, T, F, \mathcal{I}, C_{\text{in}})$ be a *pure* Non-Autonomous EN system. Let the set of transitions *potentially conflicting* with $t \in T$ be those for which there is some marking C where they are conflicting. Define two transitions t and u as being *conflicting* in C iff both are enabled and $\{t, u\}$ is *not* a step enabled at C.

Lemma 2: The set of transitions potentially conflicting with t is

$$\begin{aligned}\mathbf{conf}(t) \quad = \quad & \{u : t \neq u \wedge ({}^\bullet t \cap {}^\bullet u) \cup (t^\bullet \cap u^\bullet) - \mathfrak{I} \neq \varnothing \\ & \wedge ({}^\bullet t \cap u^\bullet) \cup (t^\bullet \cap {}^\bullet u) = \varnothing\} \end{aligned} \tag{1}$$

for all $t \in T$.

Proof: See appendix. ■

Given this notion of potential conflict, it is now possible to define an algorithm that finds a set U that is a step enabled at some configuration C. There is no attempt here to find the largest possible such set. (Note that for the set X with elements $\{x_1, x_2, \ldots, x_n\}$, $\bigvee X$ is shorthand for $x_1 \vee x_2 \vee \ldots \vee x_n$ and similarly, $\bigwedge X$ is shorthand for $x_1 \wedge x_2 \wedge \ldots \wedge x_n$.)

Theorem 1 (Step enabled at C): Let $M = (P, T, F, \mathfrak{I}, C_{\text{in}})$ be a pure Non-Autonomous EN system with a current configuration C. Let U_0 be an arbitrary subset of T. Define τ_k to be a subset of T that contains $\{t_1, t_2, \ldots, t_k\}$ for $1 \leq k \leq |T|$ and τ_0 to be the empty set. For each transition t_k being the kth element of T, if t_k is enabled at C and is not potentially conflicting with $U_{k-1} \cap \tau_{k-1}$ (i.e. $\mathbf{conf}\,(t_k) \cap U_{k-1} \cap \tau_{k-1} = \varnothing$), then $U_k = U_{k-1} \cup \{t_k\}$, otherwise $U_k = U_{k-1} - \{t_k\}$. The final set $U_{|T|}$ is a step enabled at C. If there is an enabled transition at C, then $U_{|T|}$ is non-empty.

Proof: See appendix. ■

The above theorem forms the basis for an algorithm that sequentially examines each transition and finds the set of transitions enabled at C. The set is constructed by adding in enabled transitions that are not conflicting with the ones that have been included so far. By only examining conflict with transitions that have been examined during this cycle, it is possible to avoid initialising the set U. Also, this means that if two transitions potentially conflict, only the second will perform the check for conflict, which is sensible since only the second transition ever needs to perform this check.

The set U corresponding to a step enabled at C can be represented in PLC memory by associating a unique single bit address with each transition in T. The configuration C can also be represented by mapping each place in P to a unique single bit address. Since the algorithm does not assume that the set U is initially empty, it is possible to proceed without initialising each transition bit. On this basis, the algorithm can be restated as follows.

Algorithm 1 (Find step enabled at C): Let $M = (P, T, F, \mathfrak{I}, C_{\text{in}})$ be a pure Non-Autonomous EN system with a current configuration C. Let $p_j \in P$ for $j \in [1, |P|]$ be a set of boolean variables representing membership in C. To find a step enabled at C and represent it as the ordered set of boolean variables $t_k \in T$, update (in ascending k order),

$$t_k \leftarrow \bigwedge {}^\bullet t_k \wedge \neg \bigvee t_k^\bullet \wedge \neg \bigvee (\mathbf{conf}\,(t_k) \cap \tau_{k-1}), \tag{2}$$

for all $k \in [1, |T|]$.

Remark 3: All transitions in the identified step are fired simultaneously (during the same cycle). In the case of conflict, the conflict will always be resolved in the same way (in favour of earlier transitions) given a particular ordering of transitions. Note that the ordering of transitions is arbitrary; where the choice of one transition over another is important, the designer should use sensory inputs to help determine which transition should fire, rather than to fine tune transition priorities.

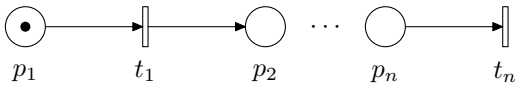

Fig. 5. Sequence of places. The marking of each place in the sequence persists for one cycle.

For example, given the net shown in figure 4, (2) gives the ordered set of updates,

$$\begin{aligned} t_1 \quad &\leftarrow \quad p_1 \wedge \neg p_3, \\ t_2 \quad &\leftarrow \quad p_2 \wedge \neg p_3 \wedge \neg t_1, \\ t_3 \quad &\leftarrow \quad p_2 \wedge \neg p_4 \wedge \neg t_2, \\ t_4 \quad &\leftarrow \quad p_5 \wedge \neg p_4. \end{aligned}$$

It is assumed that these updates are processed in the order given. Given the configuration in the diagram, t_1 and t_3 will fire.

C. Compiling Places

The second phase is to compile places to PLC instructions. Places should turn on when any transitions in its pre-set fire, turn off when any of its post-set fire, and otherwise stay in their previous state. This can readily be expressed as an algorithm for updating the configuration.

Algorithm 2 (Updating the configuration): Let $M = (P, T, F, \mathfrak{I}, C_{\text{in}})$ be a pure Non-Autonomous EN system with a current configuration C and U be a step enabled at C. Let boolean variables $p \in P$ represent membership in C, and variables $t \in T$ represent membership in U. The updated configuration C' such that $C\,[U\rangle\,C'$ corresponds to updated variables p according to the assignment,

$$p \leftarrow \bigvee {}^\bullet p \vee p \wedge \neg \bigvee p^\bullet, \tag{3}$$

for all $p \in (P - \mathfrak{I})$.

Note that input places are not updated.

Remark 4: Transient markings that are visited are always visited for at least one cycle. For example, given the EN system in figure 5, the marking at the end of the cycle is successively $\{p_2\}, \{p_3\}, \ldots, \{p_n\}, \{\}, \ldots$ with each place being visited once.

For example, applying (3) the updates for place variables for figure 4 are,

$$\begin{aligned} p_1 \quad &\leftarrow \quad p_1 \wedge \neg t_1, \\ p_2 \quad &\leftarrow \quad p_2 \wedge \neg (t_2 \vee t_3), \\ p_3 \quad &\leftarrow \quad (t_1 \vee t_2) \vee p_3 \\ p_5 \quad &\leftarrow \quad p_5 \wedge \neg t_4. \end{aligned}$$

The evaluation of these updates can occur in any order. Given the configuration $\{p_1, p_2, p_5\}$, t_1 and t_3 firing will give a new configuration of $\{p_3, p_5\}$.

D. Extending the approach to T-timed Petri nets

The compilation method described above has no timed aspect other than that induced by the implicit loop (known as the *scan cycle*) of the PLC. It is a common requirement for PLCs to have a timed interaction with the world. For example, a button may need to be depressed for a full three seconds, to ensure that it cannot be accidentally knocked, or a cooling fan may need to be run for several minutes after the cause of heat has been removed.

One approach to introduce timed interaction into the model and thus produce timed interaction behaviour in the PLC code has been developed by Jones *et al.* [22] and is based on T-timed Petri nets. The "T" stands for transition, and this means that transitions may have time delays associated with them. In Jones *et al.*'s implementation, the firing of a transition is delayed until a certain time has elapsed.

Jones *et al.* apply this approach in the context of Petri nets with no (apparent) limit on the tokens per place. If there were a limit on the number of tokens per place, it would be necessary to retest the conditions for firing when the time has elapsed to ensure that, for example, the maximum number of output tokens per place is not be exceeded (i.e. check for overflow).

Consider also the situation where two timed transitions have arcs flowing from a place with a single token. Although conflict checking might prevent them from beginning their timers simultaneously, the problem remains that both might start timers, leading both transitions to fire, and thus to the situation that one transition attempts to draw a token from an empty place (i.e. the place underflows).

The dual problems of overflow and underflow are particularly critical when dealing with Petri nets that have binary marks. They can be resolved by retesting the requirements for firing a transition after the time has elapsed, but before firing the transition. This leads to the question of what should happen if the requirements for firing a transition were only true intermittently during the timed delay. Consider, for example, a button being pressed intermittently but happening to be pressed at the start and end of the timed delay. Following the usual semantics for T-timed PNs [1, p. 98], a transition with a time delay only fires after the normal conditions for firing have occurred continuously for the specified time period.

Definition 5: A T-timed Non-Autonomous EN system is a tuple $(P, T, F, C_{\text{in}}, \mathcal{I}, W)$ where $(P, T, F, C_{\text{in}}, \mathcal{I})$ is the underlying Non-Autonomous EN system, and $W : T \to \mathbb{N}^0$ is a mapping from transitions to non-negative integers that specifies the "wait" time or time-delay associated with each transition. Transitions $t \in T$ such that $w(t) = 0$ have the usual firing semantics, whereas those transitions where $w(t) > 0$ only fire after being enabled continuously for $w(t)$ time units.

The above definition suggests that it is possible to simply extend Algorithm 1 that finds a step enabled at configuration C. It is common for PLCs to explicitly support timers, although the semantics for using them vary. Here it is assumed that an instruction and a predicate are supported: RUN_TIMER, and IS_COMPLETE. The instruction RUN_TIMER(a, b), when enabled, causes timer a to run for up to b time units. Disabling the instruction causes the timer to be reset. The predicate IS_COMPLETE(a) is true when the time has completely elapsed.

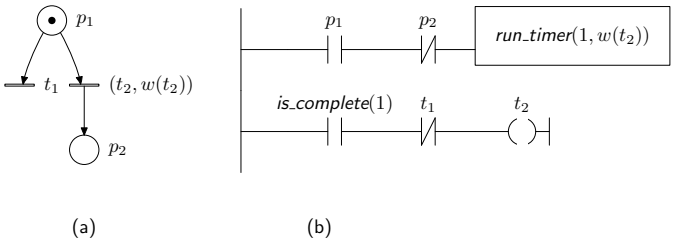

Fig. 6. T-timed Petri net (a) with a timed transition t_2 that has a time delay $w(t_2)$, and the section of ladder diagram code (b) for firing transition t_2. Note that the conflict test for t_1 only needs to be done just prior to firing the transition.

Algorithm 3 (Find step enabled with timed transitions): Let $M = (P, T, F, C_{\text{in}}, \mathcal{I}, W)$ be a pure T-timed Non-Autonomous EN system with a current configuration C. Assume the availability of a timer for each timed transition. For transitions with $w(t) = 0$, (2) applies, however when $w(t) > 0$ the following series of statements must be generated,

1) RUN_TIMER$(j, w(t_k))$ iff $\bigwedge {}^\bullet t_k \wedge \neg \bigvee t_k^\bullet$
2) $t_k \leftarrow \neg \bigvee (\mathbf{conf}\,(t_k) \cap \tau_{k-1}) \wedge$ IS_COMPLETE(j).
   for each $k \in [1, |T|]$, where j refers to the next available timer.

An example of timed interaction is shown in figure 6. For readers not familiar with the ladder diagram (LD) language, an open contact (two short vertical bars separated by a space) means take the value of the element, and a closed contact (like an open contact but with a diagonal stroke through it) means take the logical NOT of the value of the element. Horizontal line joins correspond to a logical AND, while vertical line joins correspond to a logical OR. Broken circles (referred to as coils) mean that the logical value calculated so far is output to that element. Boxes containing a function (such as "start timer" in the diagram) are only executed if the preceding logical expression evaluates to true.

IV. Discussion

An important property of (2) and (3) is that they rely only on boolean logic and assignment operations at runtime. This leads to program code that contains no conditional statements or loops. The time for each PLC scan cycle is thus $\Theta(|P|+|T|)$ for a net with $|P|$ places and $|T|$ transitions. Note also that this is a tight bound; the scan cycle time will remain constant regardless of the state of the net. This is an important property for real-time systems.

Since the statements in the resulting program are simple, diagnosis of a problem within the program is straightforward. Note that even if the program were free of bugs, diagnosis of the system as a whole, for example to find a faulty sensor, would still be important. Ladder diagram (LD) diagnosis tools are useful in this respect because they make it easy to trace the cause of this type of problem. Of course, if bugs are discovered, it is important not to change the ladder diagrams, but rather to change the Petri net and recompile it.

An issue not dealt with in this approach is that of establishing the initial configuration C_{in}. Some PLC operating systems

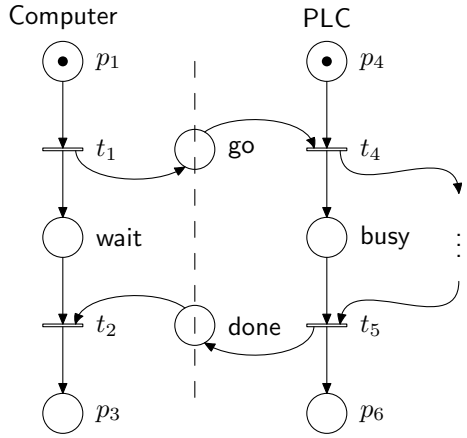

Fig. 7. Shared memory coordination pattern.

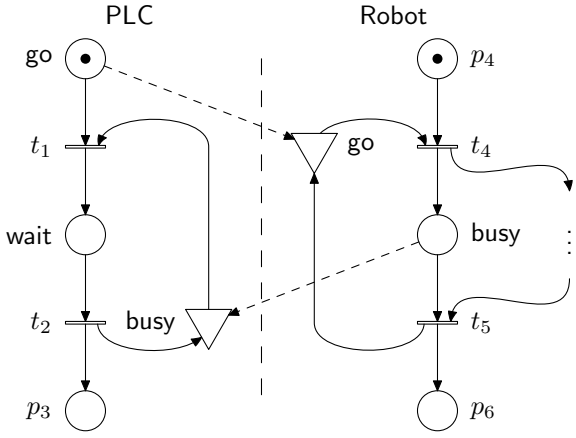

Fig. 8. Wired connection coordination pattern.

provide a "first scan cycle" flag, and this can be used to initialise the configuration. However the exact mechanism used may vary.

A. Design Patterns

A fundamental advantage of the graphical representation is that it allows commonly used patterns to be easily identified and described. To demonstrate this, two simple patterns are given, both to do with machine to machine (or module to module) communication for the purpose of coordinating behaviour. The first involves shared memory and the second performs a similar task but without requiring shared memory.

1) Shared Memory Coordination Pattern: In previous work [28], the basis for using shared memory for communicating between non-autonomous EN systems has been developed. The main outcome of this work was to show that a place could be used to communicate safely as long as there were only incoming arcs from one machine and only outgoing arcs to the other. Figure 7 shows a coordination pattern based on this approach. This is similar to Boucher's use of Petri nets for describing communication protocols [18, p. 354]. A shared memory coordination pattern is generally appropriate between a general purpose computer and PLC or between two otherwise independent modules on the same PLC. An example of the use of this pattern is given in the case study in section V.

2) Wired Connection Coordination Pattern: It is not always possible to use shared memory. Controlling a robot arm, for example, must typically be performed by wiring some of the PLC's inputs and outputs to corresponding outputs and inputs on the robot controller. Figure 8 shows a wired connection coordination pattern. In this case, no places sit on the boundary between the two machines. Instead, the PLC GO place is wired to the GO input on the robot, while the robot output BUSY is wired to the BUSY input on the PLC, as shown by the dashed arrows.

B. Modularity

There are two main requirements for modularity. First, when designing a net to control a particular device, such as the conveyor dock discussed in section V, it is useful to be able to reuse that net for as many devices as exist. Second, it is often desirable to decompose the task of controlling a device into a series of interacting sub-components to reduce the overall complexity of the design.

Module reuse is possible if modules are not hard-wired to specific addresses. In the model proposed here, module reuse is made possible by separating the definition of the structure of the net from the assignment of PLC addresses. Borrowing from the object oriented paradigm, the net is roughly like the definition of a *class*, whereas the specific use of it for a particular device is the *instance*. The graphical tool discussed in the next section supports this approach.

Task decomposition is an important tool to avoid overly complex modules. Typically, Petri net decompositional approaches use some form of hierarchy of nets and sub-nets. This is the approach taken by the SIPNEditor software [11], for example. This approach, however, can restrict the richness of the interface between sub-nets. Rather than communicating complex messages, these types of sub-nets are merely being told when to start, and then saying when they have finished.

In principle, a hierarchical approach could be used with non-autonomous EN systems. In practise, however, it was found that a different approach to task decomposition was more useful: that of using place-bordered modules. Rather than forming a hierarchy, these modules are autonomous and may initiate communication. Module to module communication can be enabled by putting one or more places on the "border" between two modules, as described in the shared memory coordination pattern described in the previous section. Potentially other communication patterns could also be used. A useful side effect of this approach is that it becomes straightforward to distribute a large system over a number of PLCs or computers by simply moving some of the modules.

C. Relationship to Function Block Architectures and Agent-based Approaches

The concept of place-bordered modules is similar to that of Function Block architectures [29]. Function blocks were originally standardised in IEC 61131-3 as one of the alternative

PLC programming languages. They have since been further developed into the IEC 61499 series of standards. IEC 61499 is a significant extension over IEC 61131-3 function blocks that emphasises software reusability, distribution, and event-based communication.

A function block typically encapsulates an algorithm, often implemented using Ladder Diagrams (LD). The algorithm could also be designed as a non-autonomous EN system module. In this way, the two approaches can be considered complementary; function blocks dealing with the higher level description of how modules plug together while the non-autonomous EN system defines how the function block responds to those signals. Indeed, the function block architecture neatly solves the problem of how to graphically specify the connections between place-bordered modules.

Vyatkin and Hanisch [30] have examined the use of Net Condition/Event Systems (NCES) as a model for IEC 61499 function blocks. In the NCES model of a distributed system, a signal event from one component can force a simultaneous action in a second component, but only if the second action is enabled. In their work they derive a single net that models a distributed system of function blocks in order to perform verification analysis. Computationally the NCES model is similar to the non-autonomous EN system. The main difference between the two being that in the modular non-autonomous EN system model presented in this paper, arcs do not cross module boundaries. Therefore, from the point of view of the PLC software developer, they can consider each module in isolation. It is interesting to note that it should be straightforward to start with a set of non-autonomous EN system modules and then compile them to a single NCES net for the purpose of using Vyatkin and Hanisch's verification approach.

The IEC 61499 architecture deals with distribution by supporting event signals into and out of a function block in addition to data flows. This more naturally supports interrupt driven or "push" style communication, whereas without event signalling, polling or "pull" style approaches will tend to be required. The place-bordered module appears initially as a polled approach. For example, t_2 in figure 7 seems to need to poll "wait" and "done" in order to find out when to fire. Nonetheless, it is possible, in the context of a general purpose computer, to only test for transitions being enabled when (a) the previous cycle caused a transition to fire, (b) the earliest timer expiry has been reached, or (c) external (input or output) places are altered. Where (c) can be supported by a hardware interrupt, the need for polling can be significantly reduced.

Several authors, including Brennan et al. [31], and Leitão et al. [32], have shown how IEC 61499 function blocks can be linked to higher-level agents. As we describe in previous work [28], the communication can be peer-to-peer in nature, allowing the low-level non-autonomous EN system to not simply respond to requests for service from high-level agents, but be able to notify them of faults or other changes in the environment asynchronously. It may seem that it would be simpler to have all components being structured as software agents, however, as Brennan et al. [31] discuss, physical machinery often requires its controller to be responsive in real-time.

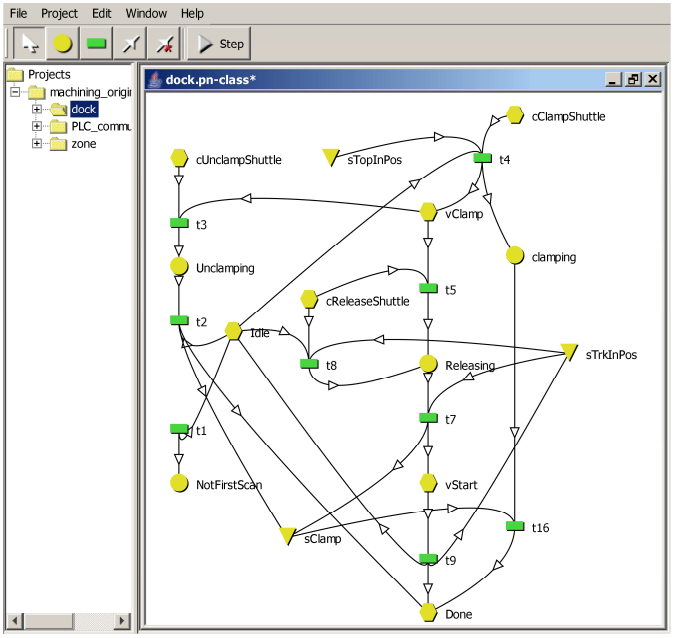

Fig. 9. PETRILLD graphical tool showing the net for a conveyor dock. In the tool, a hexagon represents a place that must have a specified address, which saves the user from having to specify addresses that do not matter.

D. Graphical Tool

To support the use of the compilation method presented in this paper, we created a graphical tool called PETRILLD[1] to allow Non-Autonomous EN systems to be designed, simulated and compiled to executable PLC program code (shown in figure 9). Note that the tool includes a hexagonally shaped place to denote a place that must have a specified address with the PLC. Circular places have their addresses dynamically assigned. The PETRILLD tool provides a number of features, including:

1) a graphical design tool that allows the user to graphically layout a Non-Autonomous EN system,
2) an address table editor that allows certain places to be associated with specific memory addresses in the PLC,
3) a simulation tool that allows places to be marked or unmarked and the firing sequence to be stepped through, and,
4) a compiler that produces PLC instructions as defined by the compilation method described in this article.

The following section makes use of this graphical development environment to develop a PLC control program for a conveyor dock.

V. CASE STUDY: CONVEYOR DOCK

To demonstrate the compilation approach, the graphical tool was used to develop PLC programs for a MonTech[2] Positioning Station (or more simply, conveyor dock), controlled by an

[1]Freely available under the GNU General Public License from `http://petrilld.sourceforge.net`

[2]See http://www.montech.de

Fig. 10. Photograph of docking station, also showing a shuttle (without a top).

Omron C200E PLC. A picture of the docking station is shown in figure 10. This docking station is one component in an experimental flexible manufacturing laboratory environment that is described in detail elsewhere [33], [34]. The conveyor track is a monorail on which shuttles move in a single direction around the track while there is nothing obstructing their path. They can also be signalled to stop and start using pneumatic signals affixed to the track. The conveyor dock is a component of this track. When a shuttle arrives at a dock, it is stopped. If it has a work-piece top, the top can be clamped in place, allowing the work-piece to be worked on. The top can then be unclamped, while leaving the shuttle stopped. This operation is useful when lifting the top off the shuttle. At any stage, the shuttle and top can be released, which causes the top to be unclamped and the shuttle to start moving along the track again.

Figure 9 shows the non-autonomous Elementary net design developed for this conveyor dock using the graphical tool. The net is drawn graphically with circles for normal places, triangles for input places, boxes for transitions and lines with arrows for directed arcs connecting places to transitions and transitions to places. In this case study, the naming convention used for valves and sensors involves a prefix of "v" or "s", respectively, followed by a descriptive name. Two output places, VCLAMP and VSTART correspond to valve actuators. The former clamps the top of the shuttle in place, while the latter signals the shuttle to start moving out of the dock. These two output places should never be active simultaneously as this might damage the shuttle motor.

The places CCLAMPSHUTTLE, CLAMPING, and DONE are an instance of the use of the shared memory coordination design pattern described in section IV-A.1.

It is often desirable to have a transition fire when a place, particularly an input place, is "false" or unmarked. For example, if a piston should fire only when a sensor detects nothing in the way. A possible approach is to augment the EN model with a "negated" place, similar to the approach used by Jones *et al.* [14]. However it is actually unnecessary to add anything to the EN model; reversing the direction of the arcs involved produces the same effect. For example, in figure 9, SCLAMP must be unmarked (i.e. false) for T7 to fire (among other conditions).

When developing the net for the conveyor dock, it was possible to test the dynamic characteristics of the net by single stepping through a simulation (using the graphical tool). This meant that most of the debugging could be performed without having to test it on the physical hardware. In principle, automated verification techniques could be used, however the tool does not currently support them. Once the net had been compiled, and the resulting code transferred to the PLC, only two problems were discovered. The first was that one of the addresses had been coded incorrectly, and the second was that commands should be normal places rather than input places (as discussed above). Previous experience with developing code by manually converting a non-autonomous Petri net model into ladder diagram code was that it generated more complicated code, and took much longer to debug.

Remark 5: In principle the SIPNEditor tool might have been used, however IEC 61131-3 IL code is not supported by the PLC being used.

A section of the resulting program, in ladder diagram form, is shown in figure 11. This section includes all statements that relate to transition T3. There is the potential for conflict with transition T5 and so the second rung shown in the figure is coded to avoid it firing simultaneously with T3. Specifically, conflict occurs if both CUNCLAMPSHUTTLE and CRELEASESHUTTLE are true, possibly indicating incorrect logic in the external computer. The conflict check ensures that even if this should happen, only one of the transitions will fire.

While the statements for places CUNCLAMPSHUTTLE, UNCLAMPING and VCLAMP have a similar structure, they correspond to different types of places. CUNCLAMPSHUTTLE corresponds to a message from a remote computer and so is only turned on by the remote computer (via the data table); UNCLAMPING corresponds to internal state and is turned on by one transition and turned off by another; and VCLAMP corresponds to a physical actuator, but is otherwise turned on and off in the usual way. This last point reinforces the idea that outputs, such as valves, lights and switches, are treated in a similar way to internal state flags. No distinction is made in the Non-Autonomous EN model, and the derived PLC program is the same for both.

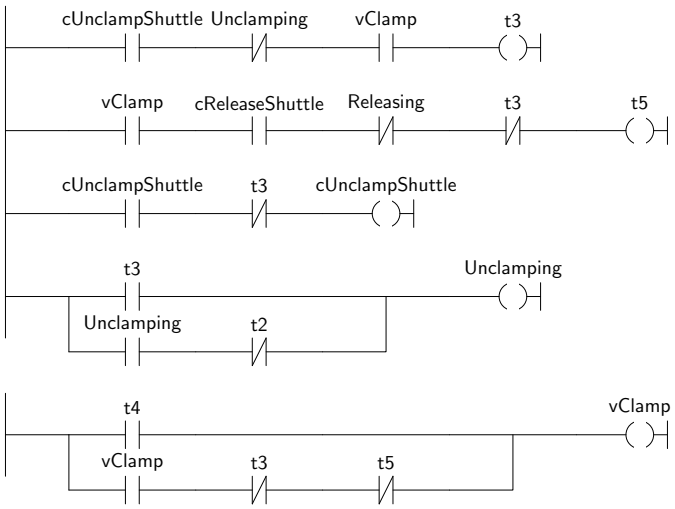

Fig. 11. A section of the program generated for the conveyor dock in ladder diagram form, showing all rungs that refer to transition T3. T5 is potentially conflicting (i.e. should not fire at the same time) and so the second rung checks to ensure that T3 is off (i.e. not firing) before firing T5.

VI. Conclusion

Manufacturers, and indeed many other industries using automation, face an increasing rate of change in their operating environment to satisfy the requirements of having an agile system. This leads to a higher rate of change being required in their automation systems. This paper contends that it is easier to cope with changes in these automation systems if it is just a matter of manipulating an abstract model, such as adding new nodes and arcs to a Petri net, rather than altering low-level program code, such as inserting new lines in a ladder diagram program. It is important as well to avoid errors when making such changes, and this can be achieved by automating the compilation process. Designing at the higher level should encourage reuse of designs and the formation of a catalogue of design patterns, since such patterns are more easily recognised when working abstractly. These factors all lead to an increase in productivity. Less skill may be required to develop correct PLC programs, and it should be quicker to do so.

This paper has introduced a formalism for the design of automation software that builds on previous work in non-autonomous Petri nets. It adds in the idea of treating inputs and outputs as first-class entities and shows how this can help with resolving ambiguity and allowing interaction patterns to be more clearly represented. The mechanism for correctly generating ladder diagram code has been derived in a way that does not require that the engineer check the net for such things as boundedness, or absence of conflict. In addition, the modelling framework provides for the reuse of modules for similar hardware components and allows the decomposition of a large net into a set of place-bordered sub-nets.

A key contribution of this work is a graphical design, simulation and compiler tool that provides the programmer with an integrated environment for developing and performing preliminary testing of PLC programs. This tool can automatically compile into a variety of output forms, including several different PLC languages and also Java, and Visual Basic. In this paper, the approach and associated tool have been demonstrated using a practical example involving a conveyor dock. Using the tool, development time was greatly reduced, and most bugs were discovered prior to testing on the hardware.

Acknowledgements

The authors would like to acknowledge the contribution of Cambridge University Manufacturing Engineering Tripos students who provided valuable suggestions and feedback about the PetriLLD tool over the last two years. The authors also wish to thank the anonymous reviewers for their helpful review comments and in particular for pointing out the relevance of IEC 61499 function blocks to this work.

References

[1] R. David and H. Alla, *Discrete, Continuous and Hybrid Petri Nets*. Springer-Verlag, 2004.

[2] J. L. Peterson, *Petri net theory and the modeling of systems*. Englewood Cliffs; London: Prentice Hall, 1981.

[3] F. DiCesare, ed., *Practice of Petri nets in manufacturing*. Chapman & Hall, 1993.

[4] K. Saitou, S. Malpathak, and H. Qvam, "Robust design of flexible manufacturing systems using, colored Petri net and genetic algorithm," *Journal of Intelligent Manufacturing*, vol. 13, pp. 339–351, 2002.

[5] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[6] J.-S. Lee and P.-L. Hsu, "An improved evaluation of ladder logic diagrams and petri nets for the sequence controller design in manufacturing systems," *Int. J. Adv. Manuf. Techol.*, vol. 24, pp. 279–287, 2004. DOI 10.1007/s00170-003-1722-y.

[7] J.-L. Chirn and D. C. McFarlane, "Petri nets based design of ladder logic diagrams," in *UKACC International Conference on CONTROL 2000*, (Cambridge, UK), September 2000.

[8] G. Frey, "Automatic implementation of Petri net based control algorithms on PLC.," in *Proc. 2000 American Control Conference*, vol. 4, (Chicago, IL), pp. 2819–2823, 28–30 June 2000.

[9] G. Frey, "PLC programming for hybrid systems via signal interpreted Petri nets," in *Proc. 4th Int. Conf. on Automation of Mixed Processes ADPM*, (Dortmund, Germany), pp. 189–194, September 18–19 2000.

[10] G. Frey and L. Litz, "Formal methods in PLC programming," in *Proc. IEEE Conference on Systems Man and Cybernetics SMC 2000*, (Nashville), pp. 2431–2436, Oct 8–11 2000.

[11] G. Frey and M. Minas, "Internet-based development of logic controllers using signal interpreted Petri nets and IEC 61131," in *Proc. 5th World Multi-Conference on Systemics, Cybernetics and Informatics (SCI 2001)*, vol. 3, (Orlando (FL) USA), pp. 297–302, July 2001.

[12] S. Klein, G. Frey, and M. Minas, "PLC programming with signal interpreted Petri nets," in *Proc. ICATPN 2003, Eindhoven (The Netherlands)*, LNCS 2679, pp. 440–449, Springer Verlag, June 2003.

[13] G. Frey, *Design and formal Analysis of Petri Net based Logic Control Algorithms*. PhD thesis, Fachbereich Elektro- und Informationstechnik der Universität Kaiserslautern, 2002.

[14] A. H. Jones, M. Uzam, A. H. Khan, D. Karimzadgan, and S. B. Kenway, "A general methodology for converting Petri nets into ladder logic: The TPLL methodology," in *Proc. 5th Int. Conf. on Computer Integrated Manufacturing and Automation Technology (CIMAT'96)*, (France), pp. 357–362, May 1996.

[15] M. Uzam and A. H. Jones, "Design of ladder logic for an agile manufacturing system using TPLL," in *Proc. 1st Turkish Symposium on Intelligent Manufacturing Systems (IMS'96)*, (Sakarya, Turkey), pp. 513–518, May 30–31 1996.

[16] M. Uzam and A. H. Jones, "Real-time implementation of Petri net controllers using programmable logic controllers," in *Proc. 4th IFAC Workshop on Algorithms and Architectures for Real-Time Control (AARTC'97)*, (Vilmoura, Portugal), pp. 421–426, April 9–11 1997.

[17] M. Uzam, *Petri-Net-based Supervisory Control of Discrete Event Systems and their Ladder Logic Diagram Implementations*. PhD thesis, Telford Research Institute, University of Salford, UK 1998.

[18] T. O. Boucher, *Computer Automation in Manufacturing: An introduction*. Chapman & Hall, 1996.

[19] G. B. Lee, H. Zandong, and J. S. Lee, "Automatic generation of ladder diagram with control Petri net," *Journal of Intelligent Manufacturing*, vol. 15, pp. 245–252, 2004.

[20] M. Minas, "SIPNeditor." Available online `http://www.eit.uni-kl.de/frey/Downloads/SIPN/SIPNEditor.htm` [Accessed 2006-09-20], October 2003.

[21] M. Minas and G. Frey, "Visual PLC-programming using signal interpreted Petri nets," in *Proc. American Control Conference 2002 (ACC2002*, pp. 5019–5024, May 2002.

[22] A. H. Jones, M. Uzam, and N. Ajlouni, "Design of discrete event control systems for programmable logic controllers using T-timed Petri nets," in *Proc. 1996 IEEE Int. Symposium on Computer-Aided Control System Design (CACSD'96)*, (Dearborn MI, USA), pp. 212–217, September 15–18 1996.

[23] K. Feldmann, A. W. Colombo, C. Schnur, and T. Stöckel, "Specification, design, and implementation of logic controllers based on colored petri net models and the standard IEC 1131 part I: Specification and design," *IEEE Trans. Control Syst. Technol.*, vol. 7, pp. 657–665, November 1999.

[24] S. S. Peng and M. C. Zhou, "Ladder diagram and Petri-net-based discrete-event control design methods," *IEEE Trans. on Syst., Man, and Cybern.*, vol. 34, pp. 523–531, November 2004.

[25] W. Reisig and G. Rozenberg, eds., *Lectures on Petri Nets I: Basic Models*, vol. 1491 of *LNCS*. Springer Verlag, 1998.

[26] L. Pomello and L. Bernadinello, "Formal tools for modular system development," in *Applications and Theory of Petri Nets 2004* (J. Cortadella, ed.), pp. 77–96, Springer, January 2004.

[27] R. W. Lewis, *Programming Industrial Control Systems Using IEC 1131-3: revised edition*. London: Institute of Electrical Engineers, 1998.

[28] J. Brusey and D. McFarlane, "Designing communication protocols for holonic control devices using elementary nets," in *Proc. 2nd Intl. Conf. on Applications of Holonic and Multi-Agent Systems (HoloMAS 2005)*, (Copenhagen, Denmark), August 2005.

[29] J. H. Christensen, "HMS/FB architecture and its implementation," in *Agent-Based Manufacturing* (S. Deen, ed.), pp. 53–88, Springer, 2003.

[30] V. Vyatkin and H.-M. Hanisch, "A modeling approach for verification of IEC1499 function blocks using net condition/event systems," in *Proc. 7th IEEE Int. Conf. Emerging Technologies and Factory Automation EFTA '99*, vol. 1, pp. 261–270, IEEE, 1999. DOI 10.1109/ETFA.1999.815365.

[31] R. Brennan, K. Hall, V. Mařík, F. Maturana, and D. Norrie, "A real-time interface for holonic control devices," in Mařík *et al.* [35], pp. 25–34.

[32] P. Leitão, R. Boissier, F. Casis, and F. Restivo, "Integration of automation resources in holonic manufacturing applications," in Mařík *et al.* [35], pp. 35–46.

[33] J. Brusey, M. Fletcher, M. Harrison, A. Thorne, S. Hodges, and D. McFarlane, "Auto-ID based control demonstration - phase 2: Pick and place packing with holonic control," tech. rep., Auto-ID Centre, Cambridge University, 2003.

[34] M. Fletcher and J. Brusey, "The story of the holonic packing cell," in *Proc. 2nd Intl. Joint Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS-03)*, (Melbourne, Australia), ACM Press, July 2003.

[35] V. Mařík, D. McFarlane, and P. Valckenaers, eds., *Holonic and Multi-Agent Systems for Manufacturing: Proc. First Intl. Conf. Industrial Applications of Holonic and Multi-Agent Systems (HoloMAS 2003)*, LNAI 2744, (Prague, Czech Republic), Springer, September 2003.

James Brusey received a B.Ap.Sci in Computer Science, and a Ph.D. from RMIT University (Melbourne, Australia) in 1996, and 2003, respectively. He is currently a Senior Research Associate with the Institute for Manufacturing in the Cambridge University Engineering Department. His current research interests include holonic or agent-based manufacturing systems, behaviour-based robotics, reinforcement learning for robotic systems, and probabilistic approaches to interpreting sensor information.

Duncan C. McFarlane is Professor of Service and Support Engineering at the Cambridge University Engineering Department, and head of the Distributed Information & Automation Laboratory within the Institute for Manufacturing. He is also Director of the Cambridge Auto-ID Lab and Research Director of two industrially supported activities: the Service and Support Engineering Programme and the Aero ID Programme. He has been involved in the design and operation of automation and information system for the manufacturing supply chain for twenty years.

Prof McFarlane completed a B Eng degree at Melbourne University in 1984, a PhD in the design of robust control systems at Cambridge in 1988, and worked industrially with BHP Australia in engineering and research positions between 1980 and 1994. Prof McFarlane joined the Department of Engineering at Cambridge in 1995 as a lecturer in the area of industrial automation systems. His research work is focused in the areas of distributed industrial automation, reconfigurable systems, RFID integration and valuing industrial information. Most recently he has been examining the role of automation and information solutions in supporting service environments.

Between 2000 and 2003 he was the European Research Director of the Auto-ID Center . In 2001 he also became a co-investigator in the EPSRC funded Innovative Manufacturing Research Centre based in the Institute for Manufacturing. In 2004, he became head of the Cambridge Auto ID Lab and co founded the Aero ID Programme, examining the role of RFID in the aerospace industry. He was appointed to the Professorship of Service and Support Engineering on 1 October 2006.

Alan Thorne graduated from Anglia Polytechnic University in Electronics and Control Systems and has a varied background in the field of Automation and Control. He has been involved in British Aerospace/IBM research projects as a systems engineer investigating flexible manufacturing systems on civil and military aircraft production. More recently he has been involved in projects relating to the development of novel AI based machine control strategies.

APPENDIX

A. Counter example for Lee et al. [19]

Consider the control PN in figure 12. Applying (4) of Lee *et al.* [19], restated here as,

$$P_i = \left(P_i + \sum_{\{t_j | t_j \in {}^\bullet p_i\}} \left(\prod_{\{p_k | p_k \in {}^\bullet t_j\}} P_k \cdot C_j \cdot E_j \right) \right) \cdot \prod_{\{t_j | t_j \in p_i^\bullet\}} \left(\overline{\prod_{\{p_k | p_k \in {}^\bullet t_j\}} P_k \cdot C_j \cdot E_j} \right)$$

yields the following update equations for p_1 and p_2:

$$\begin{aligned} p_1 &= (p_1 + p_2) \cdot \overline{p_1} \\ p_2 &= (p_2 + p_1) \cdot \overline{p_2} \end{aligned}$$

This assumes that there are no conditions or event triggers; that is, $C_j = E_j = 1$ for $j = 1, 2$. Given that the update for either p_1 or p_2 must be done first, and then the other second, it is easy to see that this causes both to become unmarked after the first iteration.

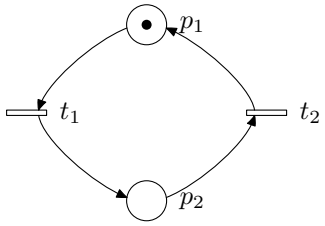

Fig. 12. Control PN for counter-example

B. Proof of Lemma 1

Proof: Given that input places $i \in \mathcal{I}$ can change state at any time, the set of configurations with the same markings over $P - \mathcal{I}$ will be strongly connected in the SCG. That is, given a particular marking for non-input places, any other marking that has the same marking for non-input places will be reachable by changing the state of some or all of the inputs. Merging each such strongly connected set of configurations and then converting the resulting SCG to an EN system yields a net with no input places. ■

C. Proof of Lemma 2

Proof: Given any element u of the set $\mathbf{conf}\,(t)$, the set $U = \{t, u\}$ must *not* conform to part (b) of the third rule of definition 3, or there must not be a marking C such that both t and u are independently enabled to fire. The first part of (1) follows directly from negating part (b) of the third rule. It remains to prove that C can exist for all t, u. Since the net is pure, both t and u must be enabled under some configuration. If t could only be enabled when u was not enabled or vice versa, then some pre-conditions of t must be post-conditions of u or some post-conditions of t must be pre-conditions of u. This is precluded by the final part of (1). ■

D. Proof of Theorem 1

Proof: First, $U_0 \cap \tau_0$ is the empty set and thus a step enabled at C, albeit a trivial one. Assuming that $U_{k-1} \cap \tau_{k-1}$ is a step enabled at C for $1 \leq k < |T|$, if t_k is enabled at C and is not potentially conflicting with $U_{k-1} \cap \tau_{k-1}$ then $(U_{k-1} \cup \{t_k\}) \cap \tau_k$ must also be a step enabled at C. If this were not the case, then t_k must conflict with a transition in $U_{k-1} \cap \tau_{k-1}$ at C. However since t_k does not *potentially* conflict with any transition in $U_{k-1} \cap \tau_{k-1}$ at any possible configuration, it cannot conflict at C. Similarly, if t_k is not enabled at C, $(U_{k-1} - \{t_k\}) \cap \tau_k$ is equivalent to $U_{k-1} \cap \tau_{k-1}$ and is thus also a step enabled at C. By induction, $U_k \cap \tau_k$ is a step enabled at C, for all $0 \leq k \leq |T|$. Therefore $U_{|T|}$ is a step enabled at C, since $U_{|T|} \cap \tau_{|T|} = U_{|T|}$. Furthermore, if there is at least one enabled transition at C, there must be at least one enabled transition t_j such that $U_{j-1} \cap \tau_{j-1}$ is empty. Since there can be no conflict with the empty set, and since no operation subsequently removes t_j, the final set $U_{|T|}$ will have at least one element. ■

E. Proof of Algorithm 1

Proof: Given some set of boolean variables V, define $\mathit{On}(V) = \{x \in V | x = \mathit{true}\}$. For any subset $X \subseteq V$, $\bigwedge X$ is true iff $X \subseteq \mathit{On}(V)$, and $\bigvee X$ is true iff $X \cap \mathit{On}(V) \neq \varnothing$. Thus (2) can be restated as,

$$\begin{aligned} t_k \quad \leftarrow \quad & ({}^\bullet t_k \subseteq C) \wedge (t_k^\bullet \cap C = \varnothing) \wedge \\ & (\mathbf{conf}\,(t_k) \cap \tau_{k-1} \cap V_{k-1} = \varnothing)\,, \end{aligned}$$

for all $k \in [1, |T|]$, where $V_k = \mathit{On}_k(T)$ is the set of true boolean variables $t_k \in T$ after the kth transition has been updated. The first part of the above expression corresponds directly to the requirements in Definition 3; that all of the pre-set of t_k and none of its post-set must be in C for t_k to be enabled to fire. The second part checks that no conflicting transition that is previous is currently firing. By its construction, the set V_k corresponds to the set U_k in Theorem 1. ■

F. Proof of Algorithm 2

Proof: Following the argument for Algorithm 1, (3) is equivalent to,

$$p \leftarrow ({}^\bullet p \cap U \neq \varnothing) \vee p \wedge (p^\bullet \cap U = \varnothing)\,,$$

for all $p \in (P - \mathcal{I})$. This can then be converted to the form,

$$p \leftarrow p \in U^\bullet \vee p \wedge \neg\,(p \in {}^\bullet U)\,,$$

for all $p \in (P - \mathcal{I})$. This update to boolean variables is equivalent to producing an updated configuration C' where,

$$C' \cap (P - \mathcal{I}) = (C \cup U^\bullet - {}^\bullet U) \cap (P - \mathcal{I})\,.$$

Since the update occurs in place, the configuration of input places is unchanged $C' \cap \mathcal{I} = C \cap \mathcal{I}$. These combine to give

$$C' = C \cup (U^\bullet - \mathcal{I}) - ({}^\bullet U - \mathcal{I})\,,$$

which can easily be shown to be the set equivalent of the second rule of Definition 3. ■

G. Proof of Algorithm 3

Proof: Part 1 ensures that the timer only runs while the transition is enabled. The timer can only be complete if it stays running for the entire time, therefore part 2 will only fire the transition if the usual firing conditions hold *and* if the transition has been enabled for $w(t)$ time units. ■